Newton Series, Coinductively

Henning Basold[1⋆], Helle Hvid Hansen[2⋆⋆], Jean-Éric Pin[3], and Jan Rutten[4,1]

[1] Radboud University Nijmegen
[2] Delft University of Technology
[3] LIAFA Université Paris VII and CNRS
[4] CWI Amsterdam

Abstract. We present a comparative study of four product operators on weighted languages: (i) the *convolution*, (ii) the *shuffle*, (iii) the *infiltration*, and (iv) the *Hadamard* product. Exploiting the fact that the set of weighted languages is a final coalgebra, we use coinduction to prove that an operator of the classical difference calculus, the *Newton transform*, generalises (from infinite sequences) to weighted languages. We show that the Newton transform is an isomorphism of rings that transforms the Hadamard product of two weighted languages into their infiltration product, and we develop various representations for the Newton transform of a language, together with concrete calculation rules for computing them.

1 Introduction

Formal languages [8] are a well-established formalism for the modelling of the behaviour of systems, typically represented by automata. *Weighted* languages – aka formal power series [3] – are a common generalisation of both formal languages (sets of words) and streams (infinite sequences). Formally, a weighted language is an assignment from words over an alphabet A to values in a set k of *weights*. Such weights can represent various things such as the multiplicity of the occurrence of a word, or its duration, or probability etc. In order to be able to add and multiply, and even subtract such weights, k is typically assumed to be a semi-ring (e.g., the Booleans) or a ring (e.g., the integers).

We present a comparative study of four product operators on weighted languages, which give us four different ways of composing the behaviour of systems. The operators under study are (i) the *convolution*, (ii) the *shuffle*, (iii) the *infiltration*, and (iv) the *Hadamard* product, representing, respectively: (i) the concatenation or sequential composition, (ii) the interleaving without synchronisation, (iii) the interleaving *with* synchronisation, and (iv) the fully synchronised interleaving, of systems. The set of weighted languages, together with the operation of *sum* and combined with any of these four product operators, is a ring itself, assuming that k is a ring. This means that in all four cases, we have a well-behaved calculus of behaviours.

⋆ Supported by NWO project 612.001.021.
⋆⋆ Supported by NWO Veni grant 639.021.231.

Main contributions: (1) We show that a classical operator from difference calculus in mathematics: the *Newton transform*, generalises (from infinite sequences) to weighted languages, and we characterise it in terms of the shuffle product. (2) Next we show that the Newton transform is an isomorphism of rings that transforms the Hadamard product of two weighted languages into an infiltration product. This allows us to switch back and forth between a fully synchronised composition of behaviours, and a shuffled, partially synchronised one. (3) We develop various representations for the Newton transform of a language, together with concrete calculation rules for computing them.

Approach: We exploit the fact that the set of weighted languages is a *final coalgebra* [16,17]. This allows us to use *coinduction* as the guiding methodology for both our definitions and proofs. More specifically, we define our operators in terms of *behavioural differential equations*, which yields, for instance, a uniform and thereby easily comparable presentation of all four product operators. Moreover, we construct *bisimulation* relations in order to prove various identities.

As the set of weighted languages over a one-letter alphabet is isomorphic to the set of streams, it turns out to be convenient to prove our results first for the special case of streams and then to generalise them to weighted languages.

Related work: The present paper fits in the coalgebraic outlook on systems behaviour, as in, for instance, [1] and [17]. The definition of Newton series for weighted languages was introduced in [14], where Mahler's theorem (which is a p-adic version of the classical Stone-Weierstrass theorem) is generalised to weighted languages. The Newton transform for streams already occurs in [12] (where it is called the discrete Taylor transform), but not its characterisation using the shuffle product, which for streams goes back to [18], and which for weighted languages is new. Related to that, we present elimination rules for (certain uses of) the shuffle product, which were known for streams [18] and are new for languages. The proof that the Newton transform for weighted languages is a ring *isomorphism* that exchanges the Hadamard product into the infiltration product, is new. In [11, Chapter 6], an operation was defined that does the reverse; it follows from our work that this operation is the inverse of the Newton transform. The infiltration product was introduced in [6]; as we already mentioned, [11, Chapter 6] studies some of its properties, using a notion of binomial coefficients for words that generalises the classical notions for numbers. The present paper introduces a new notion of binomial coefficients for words, which refines the definition of [11, Chapter 6].

Acknowledgements: We thank the anonymous referees for their constructive comments.

2 Preliminaries: stream calculus

We present basic facts from coinductive stream calculus [18]. In the following, we assume k to be a ring, unless stated otherwise. Let then the set of streams over k be given by $k^\omega = \{\, \sigma \mid \sigma : \mathbb{N} \to k \,\}$. We define the *initial value* of a stream σ by $\sigma(0)$ and its *stream derivative* by $\sigma' = (\sigma(1), \sigma(2), \sigma(3), \dots)$. In order to

conclude that two streams σ and τ are equal, it suffices to prove $\sigma(n) = \tau(n)$, for all $n \geq 0$. Sometimes this can be proved by *induction* on the natural number n but, more often than not, we will not have a succinct description or formula for $\sigma(n)$ and $\tau(n)$, and induction will be of no help. Instead, we take here a coalgebraic perspective on k^ω, and most of our proofs will use the proof principle of *coinduction*, which is based on the following notion.

A relation $R \subseteq k^\omega \times k^\omega$ is a *(stream) bisimulation* if for all $(\sigma, \tau) \in R$,

$$\sigma(0) = \tau(0) \qquad \text{and} \qquad (\sigma', \tau') \in R. \tag{1}$$

Theorem 1 (coinduction proof principle). *If there exists a bisimulation relation containing (σ, τ), then $\sigma = \tau$.*

Coinductive *definitions* are phrased in terms of stream derivatives and initial values, and are called *stream differential equations*; see [17,18,10] for examples and details.

Definition 2 (basic operators). *The following system of* stream differential equations *defines our first set of constants and operators:*

Derivative	**Initial value**	**Name**
$[r]' = [0]$	$[r](0) = r$	$r \in k$
$X' = [1]$	$X(0) = 0$	
$(\sigma + \tau)' = \sigma' + \tau'$	$(\sigma + \tau)(0) = \sigma(0) + \tau(0)$	*sum*
$(\Sigma_{i\in I}\sigma_i)' = \Sigma_{i\in I}\sigma_i'$	$(\Sigma_{i\in I}\sigma_i)(0) = \sum_{i\in I}\sigma_i(0)$	*infinite sum*
$(-\sigma)' = -(\sigma')$	$(-\sigma)(0) = -\sigma(0)$	*minus*
$(\sigma \times \tau)' = (\sigma' \times \tau) + ([\sigma(0)] \times \tau')$	$(\sigma \times \tau)(0) = \sigma(0)\tau(0)$	*convolution product*
$(\sigma^{-1})' = -[\sigma(0)^{-1}] \times \sigma' \times \sigma^{-1}$	$(\sigma^{-1})(0) = \sigma(0)^{-1}$	*convolution inverse*

The unique existence of constants and operators satisfying the equations above is ultimately due to the fact that k^ω, together with the operations of initial value and stream derivative, is a final coalgebra.

For $r \in k$, we have the constant stream $[r] = (r, 0, 0, 0, \ldots)$ which we often denote again by r. Then we have the constant stream $X = (0, 1, 0, 0, 0, \ldots)$. We define $X^0 = [1]$ and $X^{i+1} = X \times X^i$. The infinite sum $\Sigma_{i\in I}\sigma_i$ is defined only when the family $\{\sigma_i\}_{i\in I}$ is *summable*, that is, if for all $n \in \mathbb{N}$ the set $\{i \in I \mid \sigma_i(n) \neq 0\}$ is finite. If $I = \mathbb{N}$, we denote $\Sigma_{i\in I}\sigma_i$ by $\sum_{i=0}^{\infty}\sigma_i$. Note that $(\tau_i \times X^i)_i$ is summable for any sequence of streams $(\tau_i)_i$. Minus is defined only if k is a ring. In spite of its non-symmetrical definition, convolution product on streams is commutative (assuming that k is). Convolution inverse is defined for those streams σ for which the initial value $\sigma(0)$ is invertible. We will often write $r\sigma$ for $[r] \times \sigma$, and $1/\sigma$ for σ^{-1} and τ/σ for $\tau \times (1/\sigma)$, which – for streams – is equal to $(1/\sigma) \times \tau$.

The following analogue of the fundamental theorem of calculus, tells us how to compute a stream σ from its initial value $\sigma(0)$ and derivative σ'.

Theorem 3. *We have $\sigma = \sigma(0) + (X \times \sigma')$, for every $\sigma \in k^\omega$.* □

We will also use coinduction-up-to [18,15], a strengthening of the coinduction proof principle. Let us first introduce some convenient notation.

Given a relation R on k^ω, we denote by $\bar{R}$ the smallest reflexive relation on k^ω containing R and closed under the element-wise application of the operators in Definition 2. For instance, if $(\alpha, \beta), (\gamma, \delta) \in \bar{R}$ then $(\alpha + \gamma, \beta + \delta) \in \bar{R}$, etc.

A relation $R \subseteq k^\omega \times k^\omega$ is a *(stream) bisimulation-up-to* if, for all $(\sigma, \tau) \in R$,

$$\sigma(0) = \tau(0) \qquad \text{and} \qquad (\sigma', \tau') \in \bar{R}. \tag{2}$$

Theorem 4 (coinduction-up-to). *If R is a bisimulation-up-to and $(\sigma, \tau) \in R$, then $\sigma = \tau$.*

Proof. If R is a bisimulation-up-to then $\bar{R}$ can be shown to be a bisimulation relation, by structural induction on its definition. Now apply Theorem 1.

Using coinduction (up-to), one can easily prove the following.

Proposition 5 (semiring of streams – with convolution product). *If k is a semiring then the set of streams with sum and convolution product forms a semiring as well: $(k^\omega, +, [0], \times, [1])$. If k is commutative then so is k^ω.* □

Polynomial and rational streams are defined as usual, cf. [17].

Definition 6 (polynomial, rational streams). *We call a stream $\sigma \in k^\omega$* polynomial *if it is of the form $\sigma = a_0 + a_1 X + a_2 X^2 + \cdots + a_n X^n$, for $n \geq 0$ and $a_i \in k$. We call σ* rational *if it is of the form*

$$\sigma = \frac{a_0 + a_1 X + a_2 X^2 + \cdots + a_n X^n}{b_0 + b_1 X + b_2 X^2 + \cdots + b_m X^m}$$

with $n, m \geq 0$, $a_i, b_j \in k$, and b_0 is invertible.

Example 7 Here are a few concrete examples of streams (over the natural numbers): $1 + 2X + 3X^2 = (1, 2, 3, 0, 0, 0, \ldots)$, $\frac{1}{1-2X} = (2^0, 2^1, 2^2, \ldots)$, $\frac{1}{(1-X)^2} = (1, 2, 3, \ldots)$, $\frac{X}{1-X-X^2} = (0, 1, 1, 2, 3, 5, 8, \ldots)$. We note that convolution product behaves naturally, as in the following example: $(1 + 2X^2) \times (3 - X) = 3 - X + 6X^2 - 2X^3$. □

We shall be using yet another operation on streams.

Definition 8 (stream composition). *We define the composition of streams by the following stream differential equation:*

Derivative	**Initial value**	**Name**
$(\sigma \circ \tau)' = \tau' \times (\sigma' \circ \tau)$	$(\sigma \circ \tau)(0) = \sigma(0)$	*stream composition*

We will consider the composition of streams σ with τ if $\tau(0) = 0$, in which case composition enjoys the following properties.

Proposition 9 (properties of composition). *For all ρ, σ, τ with $\tau(0) = 0$, we have $[r] \circ \tau = [r]$, $X \circ \tau = \tau$, and*

$$(\rho+\sigma)\circ\tau = (\rho\circ\tau)+(\sigma\circ\tau), \quad (\rho\times\sigma)\circ\tau = (\rho\circ\tau)\times(\sigma\circ\tau), \quad \sigma^{-1}\circ\tau = (\sigma\circ\tau)^{-1}$$

and similarly for infinite sum.

Example 10 As a consequence, for rational σ, τ, the composition $\sigma \circ \tau$ amounts to replacing every X in σ by τ. For instance, $\frac{X}{1-X-X^2} \circ \frac{X}{1+X} = \frac{X(1+X)}{1+X-X^2}$. □

Defining $\sigma^{(0)} = \sigma$ and $\sigma^{(n+1)} = (\sigma^{(n)})'$, for any stream $\sigma \in k^\omega$, we have $\sigma^{(n)}(0) = \sigma(n)$. Thus $\sigma = (\sigma(0), \sigma(1), \sigma(2), \ldots) = (\sigma^{(0)}(0), \sigma^{(1)}(0), \sigma^{(2)}(0), \ldots)$. Hence every stream is equal to the stream of its *Taylor coefficients* (with respect to stream derivation). There is also the corresponding *Taylor series* representation for streams.

Theorem 11 (Taylor series). *For every $\sigma \in k^\omega$,*

$$\sigma = \sum_{i=0}^{\infty} [\sigma^{(i)}(0)] \times X^i = \sum_{i=0}^{\infty} [\sigma(i)] \times X^i$$

For some of the operations on streams, we have explicit formulae for the n-th Taylor coefficient, that is, for their value in n.

Proposition 12. *For all $\sigma, \tau \in k^\omega$, for all $n \geq 0$,*

$$(\sigma+\tau)(n) = \sigma(n)+\tau(n), \qquad (-\sigma)(n) = -\sigma(n), \qquad (\sigma\times\tau)(n) = \sum_{k=0}^{n} \sigma(k)\tau(n-k)$$

3 Four product operators

In addition to convolution product, we shall discuss also the following product operators (repeating below the definitions of convolution product and inverse).

Definition 13 (product operators). *We define four product operators by the following system of stream differential equations:*

Derivative	Initial value	Name
$(\sigma \times \tau)' = (\sigma' \times \tau) + ([\sigma(0)] \times \tau')$	$(\sigma \times \tau)(0) = \sigma(0)\tau(0)$	*convolution*
$(\sigma \otimes \tau)' = (\sigma' \otimes \tau) + (\sigma \otimes \tau')$	$(\sigma \otimes \tau)(0) = \sigma(0)\tau(0)$	*shuffle*
$(\sigma \odot \tau)' = \sigma' \odot \tau'$	$(\sigma \odot \tau)(0) = \sigma(0)\tau(0)$	*Hadamard*
$(\sigma \uparrow \tau)' = (\sigma' \uparrow \tau) + (\sigma \uparrow \tau') + (\sigma' \uparrow \tau')$	$(\sigma \uparrow \tau)(0) = \sigma(0)\tau(0)$	*infiltration*

For streams σ with invertible initial value $\sigma(0)$, we can define both convolution and shuffle inverse, as follows:

Derivative	Initial value	Name
$(\sigma^{-1})' = -[\sigma(0)^{-1}] \times \sigma' \times \sigma^{-1}$	$(\sigma^{-1})(0) = \sigma(0)^{-1}$	*convolution inverse*
$(\sigma^{-\underline{1}})' = -\sigma' \otimes \sigma^{-\underline{1}} \otimes \sigma^{-\underline{1}}$	$(\sigma^{-\underline{1}})(0) = \sigma(0)^{-1}$	*shuffle inverse*

(We will not need the inverse of the other two products.) Convolution and Hadamard product are standard operators in mathematics. Shuffle and infiltration product are, for streams, less well-known, and are better explained and understood when generalised to weighted languages, which we shall do in Section 7. Closed forms for shuffle and Hadamard are given in Prop. 15 below. In the present section and the next, we shall relate convolution product and Hadamard product to, respectively, shuffle product and infiltration product, using the so-called *Laplace* and the *Newton* transforms.

Example 14 Here are a few simple examples of streams (over the natural numbers), illustrating the differences between these four products.

$$\begin{aligned}
\frac{1}{1-X} \times \frac{1}{1-X} &= \frac{1}{(1-X)^2} = (1,\, 2,\, 3,\, \ldots) \\
\frac{1}{1-X} \otimes \frac{1}{1-X} &= \frac{1}{1-2X} = (2^0,\, 2^1,\, 2^2,\, \ldots) \\
\frac{1}{1-X} \odot \frac{1}{1-X} &= \frac{1}{1-X} \\
\frac{1}{1-X} \uparrow \frac{1}{1-X} &= \frac{1}{1-3X} = (3^0,\, 3^1,\, 3^2,\, \ldots) \\
(1-X)^{-\underline{1}} &= (0!,\, 1!,\, 2!,\, \ldots) \qquad (3)
\end{aligned}$$

We have the following closed formulae for the shuffle and Hadamard product Recall Proposition 12 for the closed form of convolution product. In Proposition 23 below, we derive a closed formula for the infiltration product as well.

Proposition 15.

$$(\sigma \otimes \tau)(n) = \sum_{i=0}^{n} \binom{n}{i} \sigma(i)\tau(n-i) \qquad (4)$$

$$(\sigma \odot \tau)(n) = \sigma(n)\tau(n) \qquad (5)$$

□

Next we consider the set of streams k^ω together with sum and, respectively, each of the four product operators.

Proposition 16 (four (semi-)rings of streams). *If k is a (semi-)ring then each of the four product operators defines a corresponding (semi-)ring structure on k^ω, as follows:*

$$\begin{aligned}
\mathcal{R}_c &= (k^\omega,\, +,\, [0],\, \times,\, [1]), & \mathcal{R}_s &= (k^\omega,\, +,\, [0],\, \otimes,\, [1]) \\
\mathcal{R}_H &= (k^\omega,\, +,\, [0],\, \odot,\, \mathsf{ones}), & \mathcal{R}_i &= (k^\omega,\, +,\, [0],\, \uparrow,\, [1])
\end{aligned}$$

where ones *denotes* $(1, 1, 1, \ldots)$. □

We recall from [12] and [18, Theorem 10.1] the following ring isomorphism between $\mathcal{R}_c$ and $\mathcal{R}_s$.

Theorem 17 (Laplace for streams, [12,18]). *Let the Laplace transform* $\Lambda : k^\omega \to k^\omega$ *be given by the following stream differential equation:*

Derivative	Initial value	Name
$(\Lambda(\sigma))' = \Lambda(d/dX(\sigma))$	$\Lambda(\sigma)(0) = \sigma(0)$	*Laplace*

where $d/dX(\sigma) = (X \otimes \sigma')' = (\sigma(1), 2\sigma(2), 3\sigma(3), \ldots)$. *Then* $\Lambda : \mathcal{R}_c \to \mathcal{R}_s$ *is an isomorphism of rings; notably, for all* $\sigma, \tau \in k^\omega$, $\Lambda(\sigma \times \tau) = \Lambda(\sigma) \otimes \Lambda(\tau)$. □

(The Laplace transform is also known as the Laplace-Carson transform.) One readily shows that $\Lambda(\sigma) = (0!\sigma(0), 1!\sigma(1), 2!\sigma(2), \ldots)$, from which it follows that Λ is bijective. Coalgebraically, Λ arises as the unique final coalgebra homomorphism between two different coalgebra structures on k^ω:

$$\begin{array}{ccc} k^\omega & \xrightarrow{\Lambda} & k^\omega \\ {\scriptstyle \langle (-)(0),\, d/dX \rangle}\downarrow & & \downarrow{\scriptstyle \langle (-)(0),\, (-)' \rangle} \\ k \times k^\omega & \xrightarrow{1 \times \Lambda} & k \times k^\omega \end{array}$$

On the right, we have the standard (final) coalgebra structure on streams, given by: $\sigma \mapsto (\sigma(0), \sigma')$, whereas on the left, the operator d/dX is used instead of stream derivative: $\sigma \mapsto (\sigma(0), d/dX(\sigma))$. The commutativity of the diagram above is precisely expressed by the stream differential equation defining Λ above. It is this definition, in terms of stream derivatives, that enables us to give an easy proof of Theorem 17, by coinduction-up-to.

As we shall see, there exists also a ring isomorphism between $\mathcal{R}_H$ and $\mathcal{R}_i$. It will be given by the *Newton* transform, which we will consider next.

4 Newton transform

Assuming that k is a ring, let the *difference operator* on a stream $\sigma \in k^\omega$ be defined by $\Delta\sigma = \sigma' - \sigma = (\sigma(1) - \sigma(0), \sigma(2) - \sigma(1), \sigma(3) - \sigma(2), \ldots)$.

Definition 18 (Newton transform). *We define the* Newton transform $\mathcal{N} : k^\omega \to k^\omega$ *by the following stream differential equation:*

Derivative	Initial value	Name
$(\mathcal{N}(\sigma))' = \mathcal{N}(\Delta\sigma)$	$\mathcal{N}(\sigma)(0) = \sigma(0)$	*Newton transform*

It follows that $\mathcal{N}(\sigma) = ((\Delta^0\sigma)(0), (\Delta^1\sigma)(0), (\Delta^2\sigma)(0), \ldots)$, where $\Delta^0\sigma = \sigma$ and $\Delta^{n+1}\sigma = \Delta(\Delta^n\sigma)$. We call $\mathcal{N}(\sigma)$ the stream of the *Newton coefficients* of σ. Coalgebraically, $\mathcal{N}$ arises as the unique mediating homomorphism – in fact, as we shall see below, an isomorphism – between the following two coalgebras:

$$\begin{array}{ccc} k^\omega & \xrightarrow{\mathcal{N}} & k^\omega \\ {\scriptstyle \langle (-)(0),\, \Delta \rangle}\downarrow & & \downarrow{\scriptstyle \langle (-)(0),\, (-)' \rangle} \\ k \times k^\omega & \xrightarrow{1 \times \mathcal{N}} & k \times k^\omega \end{array}$$

On the right, we have as before the standard (final) coalgebra structure on streams, whereas on the left, the difference operator is used instead: $\sigma \mapsto (\sigma(0), \Delta\sigma)$. We note that the term Newton transform is used in mathematical analysis [5] for an operational method for the transformation of differentiable functions. In [12], where the diagram above is discussed, our present Newton transform $\mathcal{N}$ is called the discrete Taylor transformation.

The fact that $\mathcal{N}$ is bijective follows from Theorem 20 below, which characterises $\mathcal{N}$ in terms of the shuffle product. Its proof uses the following lemma.

Lemma 19. $\frac{1}{1-X} \otimes \frac{1}{1+X} = 1$.

Note that this formula combines the convolution inverse with the shuffle product. The function $\mathcal{N}$, and its inverse, can be characterised by the following formulae.

Theorem 20 ([18]). *The function $\mathcal{N}$ is bijective and satisfies, for all $\sigma \in k^\omega$,*

$$\mathcal{N}(\sigma) = \frac{1}{1+X} \otimes \sigma, \qquad \mathcal{N}^{-1}(\sigma) = \frac{1}{1-X} \otimes \sigma.$$

At this point, we observe the following structural parallel between the Laplace transform from Theorem 17 and the Newton transform: for all $\sigma \in k^\omega$,

$$\Lambda(\sigma) = (1-X)^{-\underline{1}} \odot \sigma \tag{6}$$
$$\mathcal{N}(\sigma) = (1+X)^{-1} \otimes \sigma \tag{7}$$

The first equality is immediate from the observation that $(1-X)^{-\underline{1}} = (0!, 1!, 2!, \ldots)$. The second equality is Theorem 20.

The Newton transform is also an isomorphism of *rings*, as follows.

Theorem 21 (Newton transform as ring isomorphism). *We have that $\mathcal{N} : \mathcal{R}_H \to \mathcal{R}_i$ is an isomorphism of rings; notably, $\mathcal{N}(\sigma \odot \tau) = \mathcal{N}(\sigma) \uparrow \mathcal{N}(\tau)$, for all $\sigma, \tau \in k^\omega$.*

Expanding the definition of the shuffle product in Theorem 20, we obtain the following closed formulae.

Proposition 22. *For all $\sigma \in k^\omega$ and $n \geq 0$,*

$$\mathcal{N}(\sigma)(n) = \sum_{i=0}^{n} \binom{n}{i} (-1)^{n-i} \sigma(i), \qquad \mathcal{N}^{-1}(\sigma)(n) = \sum_{i=0}^{n} \binom{n}{i} \sigma(i)$$

From these, we can derive the announced closed formula for the infiltration product.

Proposition 23. *For all $\sigma, \tau \in k^\omega$,*

$$(\sigma \uparrow \tau)(n) = \sum_{i=0}^{n} \binom{n}{i} (-1)^{n-i} \left(\sum_{j=0}^{i} \binom{i}{j} \sigma(j) \right) \left(\sum_{l=0}^{i} \binom{i}{l} \tau(l) \right)$$

5 Calculating Newton coefficients

The Newton coefficients of a stream can be computed using the following theorem [18, Thm. 10.2(68)]. Note that the righthand side of (8) below no longer contains the shuffle product.

Theorem 24 (shuffle product elimination). *For all $\sigma \in k^\omega$, $r \in k$,*

$$\frac{1}{1-rX} \otimes \sigma = \frac{1}{1-rX} \times \left(\sigma \circ \frac{X}{1-rX}\right) \tag{8}$$

Example 25 For the Fibonacci numbers, we have

$$\mathcal{N}\,(0,1,1,2,3,5,8,\ldots) = \mathcal{N}\left(\frac{X}{1-X-X^2}\right) = \frac{X}{1+X-X^2}$$

It is immediate by Theorems 20 and 24 and Example 10 that the Newton transform preserves rationality.

Corollary 26. *A stream $\sigma \in k^\omega$ is rational iff its Newton transform $\mathcal{N}(\sigma)$ is rational.* □

6 Newton series

Theorem 20 tells us how to compute for a given stream σ the stream of its Newton coefficients $\mathcal{N}(\sigma)$, using the shuffle product. Conversely, there is the following Newton series representation, which tells us how to express a stream σ in terms of its Newton coefficients.

Theorem 27 (Newton series for streams, 1st). *For all $\sigma \in k^\omega$, $n \geq 0$,*

$$\sigma(n) \;=\; \sum_{i=0}^{n} (\Delta^i\sigma)(0)\binom{n}{i}$$

Using $\binom{n}{i} = n!/i!(n-i)!$ and writing $n^{\underline{i}} = n(n-1)(n-2)\cdots(n-i+1)$ (not to be confused with our notation for the shuffle inverse), Newton series are sometimes (cf. [9, Eqn.(5.45)]) also denoted as

$$\sigma(n) \;=\; \sum_{i=0}^{n} \frac{(\Delta^i\sigma)(0)}{i!}\, n^{\underline{i}}$$

thus emphasizing the structural analogy with Taylor series.

Combining Theorem 20 with Theorem 24 leads to yet another, and less familar expansion theorem (see [19] for a *finitary* version thereof).

Theorem 28 (Newton series for streams, 2nd; Euler expansion). *For all $\sigma \in k^\omega$,*

$$\sigma \;=\; \sum_{i=0}^{\infty} (\Delta^i\sigma)(0) \times \frac{X^i}{(1-X)^{i+1}}$$

Example 29 Theorem 28 leads, for instance, to an easy derivation of a rational expression for the stream of cubes, namely

$$(1^3, 2^3, 3^3, \ldots) = \frac{1 + 4X + X^2}{(1 - X)^4}$$ □

7 Weighted languages

Let k again be a ring or semiring and let A be a set. We consider the elements of A as *letters* and call A the *alphabet*. Let A^* denote the set of all finite sequences or *words* over A. We define the set of *languages over A with weights in k* by

$$k^{A^*} = \{\, \sigma \mid \sigma : A^* \to k \,\}$$

Weighted languages are also known as *formal power series* (over A with coefficients in k), cf. [3]. If k is the Boolean semiring $\{0, 1\}$, then weighted languages are just sets of words. If k is arbitrary again, but we restrict our alphabet to a singleton set $A = \{X\}$, then $k^{A^*} \cong k^\omega$, the set of streams with values in k. In other words, by moving from a one-letter alphabet to an arbitrary one, streams generalise to weighted languages.

From a coalgebraic perspective, much about streams holds for weighted languages as well, and typically with an almost identical formulation. This structural similarity between streams and weighted languages is due to the fact that weighted languages carry a final coalgebra structure that is very similar to that of streams, as follows. We define the *initial value* of a (weighted) language σ by $\sigma(\varepsilon)$, that is, σ applied to the *empty word* ε. Next, we define for every $a \in A$ the *a-derivative* of σ by $\sigma_a(w) = \sigma(a \cdot w)$, for every $w \in A^*$. Initial value and derivatives together define a final coalgebra structure on weighted languages, given by

$$k^{A^*} \to k \times (k^{A^*})^A \qquad \sigma \mapsto (\sigma(\varepsilon),\, \lambda a \in A.\, \sigma_a)$$

(where $(k^{A^*})^A = \{f \mid f : A \to k^{A^*}\}$). For the case that $A = \{X\}$, the coalgebra structure on the set of streams is a special case of the one above, since under the isomorphism $k^{A^*} \cong k^\omega$, we have that $\sigma(\varepsilon)$ corresponds to $\sigma(0)$, and σ_X corresponds to σ'.

We can now summarize the remainder of this paper, roughly and succinctly, as follows: if we replace in the previous sections $\sigma(0)$ by $\sigma(\varepsilon)$, and σ' by σ_a (for $a \in A$), everywhere, then most of the previous definitions and properties for streams generalise to weighted languages. Notably, we will again have a set of basic operators for weighted languages, four different product operators, four corresponding ring stuctures, and the Newton transform between the rings of Hadamard and infiltration product. (An exception to this optimistic program of translation sketched above, however, is the Laplace transform: there does not seem to exist an obvious generalisation of the Laplace transform for streams – transforming the convolution product into the shuffle product – to the corresponding rings of weighted languages.)

Let us now be more precise and discuss all of this in some detail. For a start, there is again the proof principle of coinduction, now for weighted languages.

A relation $R \subseteq k^{A^*} \times k^{A^*}$ is a *(language) bisimulation* if for all $(\sigma, \tau) \in R$:

$$\sigma(\varepsilon) = \tau(\varepsilon) \qquad \text{and} \qquad (\sigma_a, \tau_a) \in R, \text{ for all } a \in A. \tag{9}$$

We have the following *coinduction proof principle*, similar to Theorem 1:

Theorem 30 (coinduction for languages). *If there exists a (language) bisimulation relation containing (σ, τ), then $\sigma = \tau$.*

Coinductive *definitions* are given again by differential equations, now called *behavioural differential equations* [17,18].

Definition 31 (basic operators for languages). *The following system of* behavioural differential equations *defines the basic constants and operators for languages:*

Derivative	**Initial value**	**Name**
$[r]_a = [0]$	$[r](\varepsilon) = r$	$r \in k$
$b_a = [0]$	$b(\varepsilon) = 0$	$b \in A,\ b \neq a$
$b_a = [1]$	$b(\varepsilon) = 0$	$b \in A,\ b = a$
$(\sigma + \tau)_a = (\sigma_a + \tau_a)$	$(\sigma + \tau)(\varepsilon) = \sigma(\varepsilon) + \tau(\varepsilon)$	*sum*
$(\Sigma_{i\in I}\sigma_i)_a = \Sigma_{i\in I}(\sigma_i)_a$	$(\Sigma_{i\in I}\sigma_i)(\varepsilon) = \sum_{i\in I}\sigma_i(\varepsilon)$	*infinite sum*
$(-\sigma)_a = -(\sigma_a)$	$(-\sigma)(\varepsilon) = -\sigma(\varepsilon)$	*minus*
$(\sigma \times \tau)_a = (\sigma_a \times \tau) + ([\sigma(\varepsilon)] \times \tau_a)$	$(\sigma \times \tau)(\varepsilon) = \sigma(\varepsilon)\tau(\varepsilon)$	*convolution product*
$(\sigma^{-1})_a = -[\sigma(\varepsilon)^{-1}] \times \sigma_a \times \sigma^{-1}$	$(\sigma^{-1})(\varepsilon) = \sigma(\varepsilon)^{-1}$	*convolution inverse*

The convolution inverse is again defined only for σ with $\sigma(\varepsilon)$ invertible in k. We will write a both for an element of A and for the corresponding constant weighted language. We shall often use shorthands like $ab = a \times b$, where the context will determine whether a word or a language is intended. Also, we will sometimes write A for $\Sigma_{a\in A}a$. The infinite sum $\Sigma_{i\in I}\sigma_i$ is, again, only defined if the family $\{\sigma_i\}_{i\in I}$ is *summable*, i.e., if for all $w \in A^*$ the set $\{i \in I \mid \sigma_i(w) \neq 0\}$ is finite. As before, we shall often write $1/\sigma$ for σ^{-1}. Note that convolution product is weighted concatenation and is no longer commutative. As a consequence, τ/σ is now generally ambiguous as it could mean either $\tau \times \sigma^{-1}$ or $\sigma^{-1} \times \tau$. Only when the latter are equal, we shall sometimes write τ/σ. An example is $A/(1 - A)$, which is A^+, the set of all non-empty words.

Theorem 32 (fundamental theorem, for languages). *For every $\sigma \in k^{A^*}$, $\sigma = \sigma(\varepsilon) + \sum_{a\in A} a \times \sigma_a$ (cf. [7,17]).* □

We can now extend Theorem 4 to languages. Given a relation R on k^{A^*}, we denote by $\bar{R}$ the smallest reflexive relation on k^{A^*} containing R and is closed under the element-wise application of the operators in Definition 31. For instance, if $(\alpha, \beta), (\gamma, \delta) \in \bar{R}$ then $(\alpha + \gamma,\ \beta + \delta) \in \bar{R}$, etc.

A relation $R \subseteq k^{A^*} \times k^{A^*}$ is a *(weighted language) bisimulation-up-to* if for all $(\sigma, \tau) \in R$:

$$\sigma(\varepsilon) = \tau(\varepsilon), \qquad \text{and} \qquad \text{for all } a \in A : (\sigma_a, \tau_a) \in \bar{R}. \tag{10}$$

Theorem 33 (coinduction-up-to for languages). *If $(\sigma, \tau) \in R$ for some bisimulation-up-to, then $\sigma = \tau$.* □

Composition of languages is defined by the following differential equation:

Derivative	***Initial value***	***Name***
$(\sigma \circ \tau)_a = \tau_a \times (\sigma_a \circ \tau)$	$(\sigma \circ \tau)(\varepsilon) = \sigma(\varepsilon)$	composition

Language composition $\sigma \circ \tau$ is well-behaved, for arbitrary σ and τ with $\tau(\varepsilon) = 0$.

Proposition 34 (composition of languages). *For $\tau \in k^{A^*}$ with $\tau(\varepsilon) = 0$,*

$$[r] \circ \tau = [r], \qquad a \circ \tau = a \times \tau_a, \qquad A \circ \tau = \tau, \qquad \sigma^{-1} \circ \tau = (\sigma \circ \tau)^{-1}$$

$$(\rho + \sigma) \circ \tau = (\rho \circ \tau) + (\sigma \circ \tau), \qquad (\rho \times \sigma) \circ \tau = (\rho \circ \tau) \times (\sigma \circ \tau)$$

Definition 35 (polynomial, rational languages). *We call $\sigma \in k^{A^*}$* polynomial *if it can be constructed using constants ($r \in k$ and $a \in A$) and the operations of finite sum and convolution product. We call $\sigma \in k^{A^*}$* rational *if it can be constructed using constants and the operations of finite sum, convolution product and convolution inverse.* □

As a consequence of Proposition 34, for every rational σ, $\sigma \circ \tau$ is obtained by replacing every occurrence of a in σ by $a \times \tau_a$, for every $a \in A$.

Defining $\sigma_\varepsilon = \sigma$ and $\sigma_{w \cdot a} = (\sigma_w)_a$, for any language $\sigma \in k^{A^*}$, we have $\sigma_w(\varepsilon) = \sigma(w)$. This leads to a *Taylor series* representation for languages.

Theorem 36 (Taylor series, for languages). *For every $\sigma \in k^{A^*}$,*

$$\sigma = \sum_{w \in A^*} \sigma_w(\varepsilon) \times w = \sum_{w \in A^*} \sigma(w) \times w$$

Example 37 Here are a few concrete examples of weighted languages:

$$\frac{1}{1 - A} = \sum_{w \in A^*} w = A^*$$

$$\frac{1}{1 + A} = \sum_{w \in A^*} (-1)^{|w|} \times w, \qquad \frac{1}{1 - 2ab} = \sum_{i \geq 0} 2^i \times (ab)^i$$

8 Four rings of weighted languages

The definitions of the four product operators for streams generalise straightforwardly to languages, giving rise to four different ring structures on languages.

Definition 38 (product operators for languages). *We define four product operators by the following system of behavioural differential equations:*

Derivative	**Initial value**	**Name**
$(\sigma \times \tau)_a = (\sigma_a \times \tau) + ([\sigma(\varepsilon)] \times \tau_a)$	$(\sigma \times \tau)(\varepsilon) = \sigma(\varepsilon)\tau(\varepsilon)$	*convolution*
$(\sigma \otimes \tau)_a = (\sigma_a \otimes \tau) + (\sigma \otimes \tau_a)$	$(\sigma \otimes \tau)(\varepsilon) = \sigma(\varepsilon)\tau(\varepsilon)$	*shuffle*
$(\sigma \odot \tau)_a = \sigma_a \odot \tau_a$	$(\sigma \odot \tau)(\varepsilon) = \sigma(\varepsilon)\tau(\varepsilon)$	*Hadamard*
$(\sigma \uparrow \tau)_a = (\sigma_a \uparrow \tau) + (\sigma \uparrow \tau_a) + (\sigma_a \uparrow \tau_a)$	$(\sigma \uparrow \tau)(\varepsilon) = \sigma(\varepsilon)\tau(\varepsilon)$	*infiltration*

For languages σ with invertible initial value $\sigma(\varepsilon)$, we can define both convolution and shuffle inverse, as follows:

Derivative	**Initial value**	**Name**
$(\sigma^{-1})_a = -[\sigma(0)^{-1}] \times \sigma_a \times \sigma^{-1}$	$(\sigma^{-1})(0) = \sigma(0)^{-1}$	*convolution inverse*
$(\sigma^{-\underline{1}})_a = -\sigma_a \otimes \sigma^{-\underline{1}} \otimes \sigma^{-\underline{1}}$	$(\sigma^{-\underline{1}})(0) = \sigma(0)^{-1}$	*shuffle inverse*

Convolution product is concatenation of (weighted) languages and Hadamard product is the fully synchronised product, which corresponds to the intersection of weighted languages. The shuffle product generalises the definition of the shuffle operator on classical languages (over the Boolean semiring), and can be, equivalently, defined by induction. The following definition is from [11, p.126] (where shuffle product is denoted by the symbol $\circ$): for all $v, w \in A^*$, $\sigma, \tau \in k^{A^*}$,

$$v \otimes \varepsilon = \varepsilon \otimes v = v$$
$$va \otimes wb = (v \otimes wb)a + (va \otimes w)b \tag{11}$$
$$\sigma \otimes \tau = \sum_{v,w \in A^*} \sigma(v) \times \tau(w) \times (v \otimes w) \tag{12}$$

The infiltration product, originally introduced in [6], can be considered as a variation on the shuffle product that not only interleaves words but als synchronizes them on identical letters. In the differential equation for the infiltration product above, this is apparent from the presence of the additional term $\sigma_a \uparrow \tau_a$. There is also an inductive definition of the infiltration product, in [11, p.128]. It is a variant of (11) above that for the case that $a = b$ looks like

$$va \uparrow wa = (v \uparrow wa)a + (va \uparrow w)a + (v \uparrow w)a$$

However, we shall be using the coinductive definitions, as these allow us to give proofs by coinduction.

Example 39 Here are a few simple examples of weighted languages, illustrating the differences between these four products. Recall that $1/1 - A = A^*$, that is, $(1/1 - A)(w) = 1$, for all $w \in A^*$. Indicating the length of a word $w \in A^*$ by $|w|$, we have the following identities:

$$\left(\frac{1}{1-A} \times \frac{1}{1-A}\right)(w) = |w| + 1, \qquad \left(\frac{1}{1-A} \otimes \frac{1}{1-A}\right)(w) = 2^{|w|}$$

$$\frac{1}{1-A} \odot \frac{1}{1-A} = \frac{1}{1-A}, \qquad \left(\frac{1}{1-A} \uparrow \frac{1}{1-A}\right)(w) = 3^{|w|}$$

$$\left((1-A)^{-\underline{1}}\right)(w) = |w|! \tag{13}$$

If we restrict the above identities to streams, that is, if the alphabet $A = \{X\}$, then we obtain the identities on streams from Example 14. □

Next we consider the set of weighted languages together with sum and each of the four product operators.

Proposition 40 (four rings of weighted languages). *If k is a (semi-)ring then each of the four product operators defines a corresponding (semi-)ring structure on k^{A^*}, as follows:*

$$\mathcal{L}_c = \left(k^{A^*}, +, [0], \times, [1]\right), \qquad \mathcal{L}_s = \left(k^{A^*}, +, [0], \otimes, [1]\right)$$
$$\mathcal{L}_H = \left(k^{A^*}, +, [0], \odot, \frac{1}{1-A}\right), \quad \mathcal{L}_i = \left(k^{A^*}, +, [0], \uparrow, [1]\right)$$

Proof. A proof is again straightforward by coinduction-up-to, once we have adapted Theorem 33 by requiring $\bar{R}$ to be also closed under the element-wise application of all four product operators above.

We conclude the present section with closed formulae for the Taylor coefficients of the above product operators, thus generalising Propositions 12 and 15 to languages. We first introduce the following notion.

Definition 41 (binomial coefficients on words). *For all $u, v, w \in A^*$, we define $\binom{w}{u|v}$ as the number of different ways in which u can be taken out of w as a subword,* leaving *v; or equivalently – and more formally – as the number of ways in which w can be obtained by shuffling u and v; that is,*

$$\binom{w}{u \mid v} = (u \otimes v)(w) \tag{14}$$

The above definition generalises the notion of binomial coefficient for words from [11, p.121], where one defines $\binom{w}{u}$ as the number of ways in which u can be taken as a subword of w. The two notions of binomial coefficient are related by the following formula:

$$\binom{w}{u} = \sum_{v \in A^*} \binom{w}{u \mid v} \tag{15}$$

As an immediate consequence of the defining equation (14), we find the following recurrence.

Proposition 42. *For all $a \in A$ and $u, v, w \in A^*$,*

$$\binom{aw}{u \mid v} = \binom{w}{u_a \mid v} + \binom{w}{u \mid v_a} \tag{16}$$

Note that for the case of streams, (16) gives us Pascal's formula for classical binomial coefficients (by taking $a = X$, $w = X^n$, $u = X^k$ and $v = X^{n+1-k}$):

$$\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k}$$

Proposition 43 gives another property, the easy proof of which illustrates the convenience of the new definition of binomial coeficient. (It is also given in [11, Prop. 6.3.13], where $1/1 - A$ is written as A^* and convolution product as $\circ$.)

Proposition 43. *For all $u, w \in A^*$, $\left(u \otimes \frac{1}{1-A}\right)(w) = \binom{w}{u}$.*

Example 44 $\binom{abab}{ab} = \binom{abab}{ab|ab} + \binom{abab}{ab|ba} = 2 + 1 = 3$ □

We have the following closed formulae for three of our product operators.

Proposition 45. *For all $\sigma, \tau \in k^{A^*}$, $w \in A^*$,*

$$(\sigma \times \tau)(w) = \sum_{u,v \in A^* \ s.t. \ u \cdot v = w} \sigma(u)\tau(v)$$

$$(\sigma \otimes \tau)(w) = \sum_{u,v \in A^*} \binom{w}{u \mid v} \sigma(u)\tau(v) \tag{17}$$

$$(\sigma \odot \tau)(w) = \sigma(w)\tau(w) \tag{18}$$

A closed formula for the infiltration product can be derived later, once we have introduced the Newton transform for weighted languages. □

9 Newton transform for languages

Assuming again that k is a ring, we define the *difference operator* (with respect to $a \in A$) by $\Delta^a \sigma(w) = \sigma_a(w) - \sigma(w) = \sigma(a \cdot w) - \sigma(w)$, for $\sigma \in k^{A^*}$.

Definition 46 (Newton transform for languages). *We define the* Newton transform $\mathcal{N} : k^{A^*} \to k^{A^*}$ *by the following behavioural differential equation:*

Derivative	**Initial value**	**Name**
$(\mathcal{N}(\sigma))_a = \mathcal{N}(\Delta^a \sigma)$	$\mathcal{N}(\sigma)(\varepsilon) = \sigma(\varepsilon)$	*Newton transform*

(using again the symbol $\mathcal{N}$, now for weighted languages instead of streams). □

It follows that $\mathcal{N}(\sigma)(w) = (\Delta^w \sigma)(\varepsilon)$, for all $w \in A^*$, where $\Delta^\varepsilon \sigma = \sigma$ and $\Delta^{w \cdot a} \sigma = \Delta^a(\Delta^w \sigma)$. Coalgebraically, $\mathcal{N}$ arises again as a unique mediating isomorphism between two final coalgebras:

$$\begin{array}{ccc} k^{A^*} & \xrightarrow{\quad\mathcal{N}\quad} & k^{A^*} \\ \Big\downarrow{\scriptstyle \langle (-)(\varepsilon), \lambda a.\Delta^a \rangle} & & \Big\downarrow{\scriptstyle \langle (-)(\varepsilon), \lambda a.(-)_a \rangle} \\ k \times (k^{A^*})^A & \xrightarrow{1 \times \mathcal{N}} & k \times (k^{A^*})^A \end{array}$$

On the right, we have the standard (final) coalgebra structure on weighted languages, given by: $\sigma \mapsto (\sigma(\varepsilon), \lambda a \in A.\, \sigma_a)$, whereas on the left, the difference operator is used instead of the stream derivative: $\sigma \mapsto (\sigma(\varepsilon), \lambda a \in A.\, \Delta^a\sigma)$.

Theorem 47. *The function $\mathcal{N}$ is bijective and satisfies, for all $\sigma \in k^{A^*}$,*

$$\mathcal{N}(\sigma) = \frac{1}{1+A} \otimes \sigma, \qquad \mathcal{N}^{-1}(\sigma) = \frac{1}{1-A} \otimes \sigma$$

(Note again that these formulae combine the convolution inverse with the shuffle product.) The Newton transform is again an isomorphism of *rings*.

Theorem 48 (Newton transform as ring isomorphism for languages). *The Newton transform $\mathcal{N} : \mathcal{L}_H \to \mathcal{L}_i$ is an isomorphism of rings; notably, $\mathcal{N}(\sigma \odot \tau) = \mathcal{N}(\sigma) \uparrow \mathcal{N}(\tau)$, for all $\sigma, \tau \in k^\omega$.*

Noting that $\mathcal{N}(\frac{1}{1-A}) = [1]$, a proof of the theorem by coinduction-up to is straightforward. Part of this theorem is already known in the literature: [11, Theorem 6.3.18] expresses (for the case that $k = \mathbb{Z}$) that $\frac{1}{1-A} \otimes (-)$ transforms the infiltration product of two words into a Hadamard product.

Propositions 22 and 23 for streams straightforwardly generalise to weighted languages. Also Theorem 24 generalises to weighted languages, as follows.

Theorem 49 (shuffle product elimination for languages). *For all $\sigma \in k^{A^*}$, $r \in k$,*

$$\frac{1}{1-(r \times A)} \otimes \sigma = \frac{1}{1-(r \times A)} \times \left(\sigma \circ \frac{A}{1-(r \times A)}\right) \tag{19}$$

Corollary 50. *For all $\sigma \in k^{A^*}$, σ is rational iff $\mathcal{N}(\sigma)$ is rational. For all $\sigma, \tau \in k^{A^*}$, if both $\mathcal{N}(\sigma)$ and $\mathcal{N}(\tau)$ are polynomial resp. rational, then so is $\mathcal{N}(\sigma \odot \tau)$.*

Example 51 We illustrate the use of Theorem 49 in the calculation of the Newton transform with an example, stemming from [14, Example 2.1]. Let $A = \{\hat{0}, \hat{1}\}$, where we use the little festive hats to distinguish these alphabet symbols from $0, 1 \in k$. We define $\beta \in k^{A^*}$ by the following behavioural differential equation: $\beta_{\hat{0}} = 2 \times \beta$, $\beta_{\hat{1}} = (2 \times \beta) + \frac{1}{1-A}$, $\beta(\varepsilon) = 0$. Using Theorem 32, we can solve the differential equation above, and obtain the following expression: $\beta = \frac{1}{1-2A} \times \hat{1} \times \frac{1}{1-A}$. We have, for instance, that $\beta(\hat{0}\hat{1}\hat{1}) = \beta_{\hat{0}\hat{1}\hat{1}}(\varepsilon) = \left((8 \times \beta) + \frac{6}{1-A}\right)(\varepsilon) = 6$. More generally, β assigns to each word in A^* its value as a binary number (least significant digit first). By an easy computation, we find: $\mathcal{N}(\beta) = \frac{1}{1-A} \times \hat{1}$; in other words, $\mathcal{N}(\beta)(w) = 1$, for all w ending in $\hat{1}$. □

10 Newton series for languages

Theorem 27 generalises to weighted languages as follows.

Theorem 52 (Newton series for languages, 1st). *For all* $\sigma \in k^{A^*}$, $w \in A^*$,

$$\sigma(w) = \sum_u \binom{w}{u} (\Delta^u \sigma)(\varepsilon)$$

Also Theorem 28 generalises to weighted languages.

Theorem 53 (Newton series for languages, 2nd; Euler expansion). *For all* $\sigma \in k^{A^*}$,

$$\sigma = \sum_{a_1 \cdots a_n \in A^*} (\Delta^{a_1 \cdots a_n} \sigma)(\varepsilon) \times \frac{1}{1-A} \times a_1 \times \frac{1}{1-A} \times \cdots \times a_n \times \frac{1}{1-A}$$

where we understand this sum to include $\sigma(\varepsilon) \times \frac{1}{1-A}$, corresponding to $\varepsilon \in A^*$.

11 Discussion

All our definitions are coinductive, given by behavioural differential equations, allowing all our proofs to be coinductive as well, that is, based on constructions of bisimulation (up-to) relations. This makes all proofs uniform and transparent. Moreover, coinductive proofs can be easily automated and often lead to efficient algorithms, for instance, as in [4]. There are several topics for further research: (i) *Theorems 52 and 53* are pretty but are they also useful? We should like to investigate possible applications. (ii) The *infiltration product* deserves further study (including its restriction to streams, which seems to be new). It is reminiscent of certain versions of synchronised merge in process algebra (cf. [2]), but it does not seem to have ever been studied there. (iii) *Theorem 47* characterises the Newton transform in terms of the shuffle product, from which many subsequent results follow. Recently [13], Newton series have been defined for functions from words to words. We are interested to see whether our present approach could be extended to those as well. (iv) *Behavioural differential equations* give rise to weighted automata (by what could be called the 'splitting' of derivatives into their summands, cf. [10]). We should like to investigate whether our representation results for Newton series could be made relevant for weighted automata as well. (v) Our new *Definition 41* of binomial coefficients for words, which seems to offer a precise generalisation of the standard notion for numbers and, e.g., Pascal's formula, deserves further study.

References

1. Barbosa, L.: Components as Coalgebras. Ph.D. thesis, Universidade do Minho, Braga, Portugal (2001)
2. Bergstra, J., Klop, J.W.: Process algebra for synchronous communication. Information and control 60(1), 109–137 (1984)
3. Berstel, J., Reutenauer, C.: Rational series and their languages, EATCS Monographs on Theoretical Computer Science, vol. 12. Springer-Verlag (1988)

4. Bonchi, F., Pous, D.: Hacking nondeterminism with induction and coinduction. Commun. ACM 58(2), 87–95 (2015)
5. Burns, S.A., Palmore, J.I.: The newton transform: An operational method for constructing integral of dynamical systems. Physica D: Nonlinear Phenomena 37(13), 83 – 90 (1989)
6. Chen, K., Fox, R., Lyndon, R.: Free differential calculus, IV - The quotient groups of the lower series. Annals of Mathemathics. Second series 68(1), 81–95 (1958)
7. Conway, J.: Regular algebra and finite machines. Chapman and Hall (1971)
8. Eilenberg, S.: Automata, languages and machines (Vol. A). Pure and Applied Mathematics, Academic Press (1974)
9. Graham, R., Knuth, D., Patashnik, O.: Concrete mathematics (second edition). Addison-Wesley (1994)
10. Hansen, H.H., Kupke, C., Rutten, J.: Stream differential equations: specification formats and solution methods. Report FM-1404, CWI (2014), available at URL: www.cwi.nl.
11. Lothaire, M.: Combinatorics on words. Cambridge Mathematical Library, Cambridge University Press (1997)
12. Pavlović, D., Escardó, M.: Calculus in coinductive form. In: Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science. pp. 408–417. IEEE Computer Society Press (1998)
13. Pin, J.E.: Newton's forward difference equation for functions from words to words, to appear in Springer's LNCS, 2015.
14. Pin, J.E., Silva, P.V.: A noncommutative extension of Mahler's theorem on interpolation series. European Journal of Combinatorics 36, 564–578 (2014)
15. Rot, J., Bonsangue, M., Rutten, J.: Coalgebraic bisimulation-up-to. In: SOFSEM. LNCS, vol. 7741, pp. 369–381. Springer (2013)
16. Rutten, J.: Universal coalgebra: a theory of systems. Theoretical Computer Science 249(1), 3–80 (2000), fundamental Study.
17. Rutten, J.: Behavioural differential equations: a coinductive calculus of streams, automata, and power series. Theoretical Computer Science 308(1), 1–53 (2003), fundamental Study.
18. Rutten, J.: A coinductive calculus of streams. Mathematical Structures in Computer Science 15, 93–147 (2005)
19. Scheid, F.: Theory and problems of numerical analysis (Schaum's outline series). McGraw-Hill (1968)